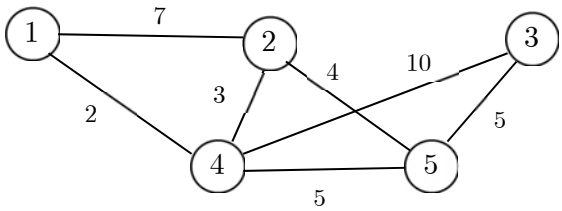
Shortest Path Algorithm

Shortest Path Algorithm เป็นขั้นตอนวิธีในการหาเส้นทางที่สั้นที่สุดจากโหนดหนึ่งไปยังโหนดอื่น ๆ (single-source shortest paths problem) โดยระยะทางจะคำนวณจากผลรวมของน้ำหนักในเส้นเชื่อมแต่ละเส้นของเส้นทางรวมกัน เช่น



จากกราฟข้างต้นการเดินทางจากโหนด 1 ไปยังโหนด 5 มีได้หลายเส้นทาง เช่น

- โหนด 1 --> โหนด 2 --> โหนด 5 มีระยะทางรวมเท่ากับ 11
- โหนด 1 --> โหนด 4 --> โหนด 5 มีระยะทางรวมเท่ากับ 7
- โหนด 1 --> โหนด 2 --> โหนด 4 --> โหนด 5 มีระยะทางรวมเท่ากับ 15
- โหนด 1 --> โหนด 4 --> โหนด 2 --> โหนด 5 มีระยะทางรวมเท่ากับ 9
- โหนด 1 --> โหนด 4 --> โหนด 3 --> โหนด 5 มีระยะทางรวมเท่ากับ 17

จากเส้นทางดังกล่าวจะได้ว่าเส้นทางที่สั้นที่สุดจากโหนด 1 ไปยังโหนด 5 จะเป็นเส้นทางจากโหนด 1 ไปยังโหนด 4 และไปโหนด 5 ซึ่งมีระยะทางรวมเท่ากับ 7

การประยุกต์ใช้งานเราสามารถย่อส่วนปัญหาในชีวิตจริงให้เป็นปัญหาทางคณิตศาสตร์และแก้ปัญหาด้วยการใช้ขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุดได้ เช่น การให้จุดยอดเป็นเมืองและเส้นเชื่อมเป็นถนน เพื่อหาเส้นทางในการเดินทางจากเมืองหนึ่งไปยังอีกเมืองหนึ่ง เพื่อให้ประหยัดเวลาหรือค่าใช้จ่ายน้อยที่สุด

คำถาม

ประเทศแห่งหนึ่งมีลักษณะเป็นหมู่เกาะจำนวน 5 เกาะ ซึ่งแต่ละเกาะจะมีชื่อเรียกแตกต่างกันและมีระยะห่างระหว่างเกาะไม่เท่ากัน เมื่อมีระบบไฟฟ้าเข้าในประเทศประธานาธิบดีต้องการเชื่อมต่อระบบไฟฟ้าจากเกาะหมายเลข 1 ซึ่งเป็นเมืองหลวงไปถึงยังเกาะหมายเลข 5 ซึ่งเป็นเมืองท่องเที่ยว แต่เนื่องด้วยท่านประธานาธิบดีต้องการประหยัดงบในการจัดซื้อสายไฟฟ้าจึงขอให้ท่านช่วยคิดว่าจะต้องเชื่อมโยงสายไฟอย่างไรที่ทำให้ประหยัดที่สุด โดยการเชื่อมโยงจะต้องมีการเชื่อมผ่านเกาะ

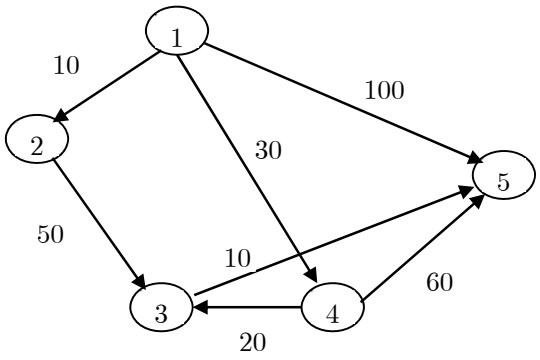
ระยะทางระหว่างเกาะ 1 กับเกาะ 2 3 และ 4 เท่ากับ 20 10 และ 30 กิโลเมตรตามลำดับ
ระยะทางระหว่างเกาะ 2 กับเกาะ 3 4 และ 5 เท่ากับ 40 35 และ 45 กิโลเมตรตามลำดับ
ระยะทางระหว่างเกาะ 3 กับเกาะ 4 และ 5 เท่ากับ 35 และ 45 กิโลเมตรตามลำดับ
ระยะทางระหว่างเกาะ 4 กับเกาะ 5 เท่ากับ 15 กิโลเมตร

Dijkstra Algorithm

ขั้นตอนวิธีของ Dijkstra ใช้แก้ปัญหาวิถีสั้นสุดแบบแหล่งต้นทางเดียว โดยที่น้ำหนักของเส้นเชื่อมต้องไม่เป็นลบ ซึ่งมีจำนวนทั้งหมด n โหนด

```
1    Procedure Dijkstra;
2    {Dijkstra computes the cost of shortest paths from vertex 1 to every
3    vertex of a directed graph}
4    begin
5         S := {1}
6         for i = 2 to n do
7             D[i] := C[1,i]
8         end
9         for i := 1 to n-1 do
10            choose a vertex w in V-S such that D[w] is a minimum;
11            add w to S
12            for each vertex v in V-S do
13                D[v] := min(D[v], D[w] + C[w,v])
14             end
15        end
16   end   //Dijkstra procedure
17
```

ตัวอย่าง การหาระยะทางที่สั้นที่สุดจากโหนด 1 ไปยังโหนด 5 ด้วยวิธีของ Dijkstra Algorithm



วิธีทำ

โดย V เป็นเซตของโหนดทั้งหมด

- เลือกโหนด 1 เป็นโหนดเริ่มต้น และสร้างเซต S โดยมีโหนดเริ่มต้นเป็นสมาชิก
- กำหนดค่าเริ่มของ D[i] ซึ่งเป็นค่าน้ำหนักจากโหนด 1 ไปยังโหนด i ใด ๆ

S = {1} L.5

iteration	S	w	D(2)	D(3)	D(4)	D(5)
0	{1}	-	10	∞	30	100

c(1,2) c(1,3) c(1,4)

{2, 3, 4, 5}

- เลือกโหนด w ในเซต V-S เพื่อเพิ่มเข้าไปในเซต S ในที่นี้เลือกโหนด 2 เนื่องจาก D[2] = 10, D[3] = ∞, D[4] = 30, D[5] = 100 (เลือกที่มีค่าน้อยที่สุด)
- คำนวณค่า D[v] ของแต่ละโหนด v ในเซต V-S ด้วย

D[v] := min(D[v], D[w] + C[w,v])

จะได้ว่า

D[3] = min(D[3], D[2]+C[2,3]) = min(∞ , 10+50) = 60

D[4] = min(D[4], D[2]+C[2,4]) = min(30 , 10+∞) = 30

D[5] = min(D[5], D[2]+C[2,5]) = min(100 , 10+∞) = 100

iteration	S	w	D(2)	D(3)	D(4)	D(5)
0	{1}	-	10	∞	30	100
1	{1,**2**}	2	**10**	**60**	**30**	**100**

- เลือกโหนด w ในเซต V-S เพื่อเพิ่มเข้าไปในเซต S ในที่นี้เลือกโหนด 4 เนื่องจาก D[3] = 60, D[4] = 30, D[5] = 100 (เลือกที่มีค่าน้อยที่สุด)
- คำนวณค่า D[v] ของแต่ละโหนด v ในเซต V-S ด้วย

D[v] := min(D[v], D[w] + C[w,v])

จะได้ว่า

D[3] = min(D[3], D[4]+C[4,3]) = min(60 , 30+20) = 50

D[5] = min(D[5], D[4]+C[4,5]) = min(100 , 30+60) = 90

iteration	S	w	D(2)	D(3)	D(4)	D(5)
0	{1}	-	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,**4**}	4	**10**	**50**	**30**	**90**

- เลือกโหนด w ในเซต V-S เพื่อเพิ่มเข้าไปในเซต S ในที่นี้เลือกโหนด 4 เนื่องจาก D[3] = 50, D[5] = 90 (เลือกที่มีค่าน้อยที่สุด)
- คำนวณค่า D[v] ของแต่ละโหนด v ในเซต V-S ด้วย

D[v] := min(D[v], D[w] + C[w,v])

จะได้ว่า

D[5] = min(D[5], D[3]+C[3,5]) = min(90, 50+10) = 60

iteration	S	w	D(2)	D(3)	D(4)	D(5)
0	{1}	-	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,**3**)	3	**10**	**50**	**30**	**90**

- เลือกโหนด w ในเซต V-S เพื่อเพิ่มเข้าไปในเซต S ในที่นี้เลือกโหนด 5 เข้าเซต S เนื่องจาก D[5] = 60 (เลือกที่มีค่าน้อยที่สุด)
- คำนวณค่า D[v] ของแต่ละโหนด v ในเซต V-S ด้วย

  D[v] := min(D[v], D[w] + C[w,v])

จะได้ว่า

iteration	S	w	D(2)	D(3)	D(4)	D(5)
0	{1}	-	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3)	3	10	50	30	60
4	{1,2,4,3,**5**}	5	**10**	**50**	**30**	**60**

สรุปเส้นทางที่สั้นที่สุดจากโหนด 1 ไปยังโหนด 5 เท่ากับ 60

Implementation:

Assume the source vertex = 0.

```
1   // A C program for Dijkstra's single source shortest path algorithm.
2
3   #include <limits.h>
4   #include <stdio.h>
5   #include <stdbool.h>
6
7   #define V 9
8
9   int minDistance(int dist[], bool sptSet[])
10  {
11      int min = INT_MAX, min_index;
12
13      for (int v = 1; v < V; v++)
14          if (sptSet[v] == false && dist[v] <= min)
15              min = dist[v], min_index = v;
16
17      return min_index;
18  }
19
20  void printSolution(int dist[])
21  {
22      printf("Vertex \t\t Distance from Source\n");
23      for (int i = 0; i < V; i++)
24          printf("%d \t\t %d\n", i, dist[i]);
25  }
26
27
28
29
30
31
32
33
34
```

```
35
36  void dijkstra(int graph[V][V], int src)
37  {
38      int dist[V];
39
40      bool sptSet[V];
41      for (int i = 0; i < V; i++) {
42          if (graph[0][i] != 0)
43              dist[i] = graph[0][i];
44          else
45              dist[i] = INT_MAX;
46          sptSet[i] = false;
47      }
48      sptSet[0] = true;
49      dist[0]= 0;
50      // Find shortest path for all vertices
51      for (int count = 0; count < V - 1; count++) {
52          int u = minDistance(dist, sptSet);
53
54         // Mark the picked vertex as processed
55         sptSet[u] = true;
56
57         // Update dist value of the adjacent vertices of the picked vertex.
58          for (int v = 1; v < V; v++) {
59              if (dist[u] != INT_MAX && graph[u][v] != 0 &&
60                  dist[u] + graph[u][v] < dist[v] )
61                  dist[v] = dist[u] + graph[u][v];
63          }
64      }
65
66      // print the constructed distance array
67      printSolution(dist);
68  }
69
70  // driver program to test above function
71  int main()
72  {
73      /* Let us create the example graph discussed above */
74      int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
75                          { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
76                          { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
77                          { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
78                          { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
79                          { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
80                          { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
81                          { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
82                          { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
83
84      dijkstra(graph, 0);
85      return 0;
86  }
87
88
89
90
```

Implementation:

Assume the source vertex = 1.

```
#define SIZE 100000 + 1

// each vertex has all the connected vertices with the edges weights
vector < pair < int , int > > v [SIZE];
int dist [SIZE];
bool vis [SIZE];

void dijkstra(){

// set the vertices distances as infinity
// set all vertex as unvisited
// multiset do the job as a min-priority queue
memset(vis, false , sizeof vis);
dist[1] = 0;
multiset < pair < int , int > > s;

// insert the source node with distance = 0
s.insert({0 , 1});

while(!s.empty()){

   // pop the vertex with the minimum distance
   pair <int , int> p = *s.begin();
   s.erase(s.begin());

   int x = p.s;
   int wei = p.f;
   // check if the popped vertex is visited before
   if( vis[x] ) continue;
   vis[x] = true;

    for(int i = 0; i < v[x].size(); i++){
         int e = v[x][i].s; int w = v[x][i].f;
         // check if the next vertex distance could be minimized
         if(dist[x] + w < dist[e]  ){
              dist[e] = dist[x] + w;
              // insert the next vertex with the updated distance
              s.insert({dist[e],  e} );
         }
     }
   }
}
```

https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/

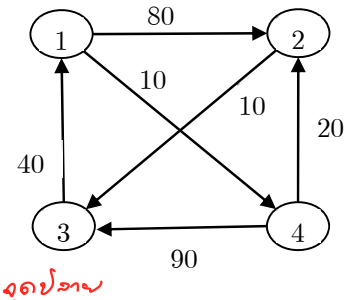
Floyd's Algorithm

Floyd's Algorithm เป็นการหาระยะทางของเส้นทางที่สั้นที่สุดระหว่างโหนดทุก ๆ คู่ (all - pairs shortest paths (APSP) problem) โดยใช้วิธีการของ Floyd มีขั้นตอนการทำงานดังนี้

```
1   Procedure Floyd-Warshall algorithm;
2   begin
3        for each node v in V do
4              Distance[v][v] := 0;
5        end
6        for each eage(s,p) in E do
7              Distance[s][p] := weight(s,p);
8        end
9        for k = 1 to n do
10            for i = 1 to n do
11                 for j = 1 to n do
12                      if Distance[i][j] > Distance[i][k]+Distance[k][j] then
13                           Distance[i][j] = Distance[i][k]+Distance[k][j]
14                      end
15                 end
16            end
17       end
18  end procedure
19
20
```

Time Complexity: $O(V^3)$

ตัวอย่าง การหาระยะทางที่สั้นที่สุดจากโหนด 1 ไปยังโหนด 5 ด้วยวิธีของ Floyd's Algorithm



n = 4

V = {1, 2, 3, 4}

Line D[v][v] = 0

กำหนดค่าเริ่มต้นของตาราง Distance

โหนด	1	2	3	4
1	0	80	∞	10
2	∞	0	10	∞
3	40	∞	0	∞
4	∞	20	90	0

เมื่อ **k =1** พิจารณา Update ตาราง Distance ด้วยเงื่อนไขต่อไปนี้

```
if Distance[i][j] > Distance[i][k]+Distance[k][j] then
    Distance[i][j] = Distance[i][k]+Distance[k][j]
end
```

เช่น Distance[3][2] = ∞, Distance[3][1] = 40, Distance[1][2] = 80 จะได้ Distance[3][2] = 120

Distance[3][4] = ∞, Distance[3][1] = 40, Distance[1][4] = 10 จะได้ Distance[3][4] = 50

โหนด	**1**	**2**	**3**	**4**
1	0	80	∞	10
2	∞	0	10	∞
3	40	120	0	50
4	∞	20	90	0

เมื่อ k = 2 พิจารณา Update ตาราง Distance ด้วยเงื่อนไขต่อไปนี้ จะได้ดังตารางข้างล่าง

```
if Distance[i][j] > Distance[i][k]+Distance[k][j] then
    Distance[i][j] = Distance[i][k]+Distance[k][j]
end
```

เช่น Distance[1][3] = ∞, Distance[1][2] = 80, Distance[2][3] = 10 จะได้ Distance[1][3] = 90

Distance[4][3] = 90, Distance[4][2] = 20, Distance[2][3] = 10 จะได้ Distance[4][3] = 30

โหนด	**1**	**2**	**3**	**4**
1	0	80	90	10
2	∞	0	10	∞
3	40	120	0	50
4	∞	20	30	0

เมื่อ k = 3 พิจารณา Update ตาราง Distance ด้วยเงื่อนไขต่อไปนี้ จะได้ดังตารางข้างล่าง

```
if Distance[i][j] > Distance[i][k]+Distance[k][j] then
    Distance[i][j] = Distance[i][k]+Distance[k][j]
end
```

เช่น Distance[2][1] = ∞, Distance[2][3] = 10, Distance[3][1] = 40 จะได้ Distance[2][1] = 50

Distance[2][4] = ∞, Distance[2][3] = 10, Distance[3][4] = 50 จะได้ Distance[2][4] = 60

Distance[4][1] = ∞, Distance[4][3] = 30, Distance[3][1] = 40 จะได้ Distance[4][1] = 70

โหนด	1	2	3	4
1	0	80	90	10
2	50	0	10	60
3	40	120	0	50
4	70	20	30	0

เมื่อ k = 4 พิจารณา Update ตาราง Distance ด้วยเงื่อนไขต่อไปนี้ จะได้ดังตารางข้างล่าง

```
if Distance[i][j] > Distance[i][k]+Distance[k][j] then
    Distance[i][j] = Distance[i][k]+Distance[k][j]
end
```

เช่น Distance[1][2] = 80, Distance[1][4] = 10, Distance[4][2] = 20 จะได้ Distance[1][2] = 30
Distance[1][3] = 90, Distance[1][4] = 10, Distance[4][3] = 30 จะได้ Distance[1][3] = 40
Distance[3][2] = 120, Distance[3][4] = 50, Distance[4][2] = 20 จะได้ Distance[3][2] = 70

โหนด	1	2	3	4
1	0	30	40	10
2	50	0	10	60
3	40	70	0	50
4	70	20	30	0

```c
// Floyd-Warshall Algorithm in C

#include <stdio.h>

// defining the number of vertices
#define nV 4

#define INF 999

void printMatrix(int matrix[][nV]);

// Implementing floyd warshall algorithm
void floydWarshall(int graph[][nV]) {
  int matrix[nV][nV], i, j, k;

  for (i = 0; i < nV; i++)
    for (j = 0; j < nV; j++)
      matrix[i][j] = graph[i][j];

  // Adding vertices individually
  for (k = 0; k < nV; k++) {
    for (i = 0; i < nV; i++) {
      for (j = 0; j < nV; j++) {
        if (matrix[i][k] + matrix[k][j] < matrix[i][j])
          matrix[i][j] = matrix[i][k] + matrix[k][j];
      }
    }
  }
  printMatrix(matrix);
}

void printMatrix(int matrix[][nV]) {
  for (int i = 0; i < nV; i++) {
    for (int j = 0; j < nV; j++) {
      if (matrix[i][j] == INF)
        printf("%4s", "INF");
      else
        printf("%4d", matrix[i][j]);
    }
    printf("\n");
  }
}

int main() {
  int graph[nV][nV] = {{0, 3, INF, 5},
                {2, 0, INF, 4},
                {INF, 1, 0, INF},
                {INF, INF, 2, 0}};
  floydWarshall(graph);
}

```

https://www.programiz.com/dsa/floyd-warshall-algorithm